A feature might be in either the 'Needs validation', 'Allowed' or 'Disallowed' state. The significance of these states are as follows:

1. Needs validation
    a. The feasibility of using the feature in CRYENGINE's code base is yet to be validated by the CRYENGINE Technical Council.
2. Allowed
    a. The feature doesn't introduce 'show-blocking' issues.
    b. The feature improves, or at least doesn't harm, code runtime and compile time performance.
    c. The feature improves, or at least doesn't harm, code readability or debugability.
    d. The feature is supported by all the compilers that the CRYENGINE code base is compiled against.
3. Disallowed
    a. The feature introduces a 'show-blocking' issue.
    b. The feature has a negative impact on runtime and compile time performance.
    c. The feature has a negative impact on code readability and debugability.
    d. The feature generally doesn't contribute to the overall quality and performance of the CRYENGINE code base.
    e. The feature is not supported by a CRYENGINE code base compiler.

Feature	State	Comments
Template argument deduction for class templates	Needs validation	-
Declaring non-type template parameters with auto	Needs validation	-
Folding expressions	Needs validation	-
New rules for auto deduction from braced-init-list	Needs validation	-
Constexpr lambda	Needs validation	-
Lambda capture this by value	Needs validation	-
Inline variables	Needs validation	-
Nested namespaces	Allowed	This feature is considered light weight and doesn't negatively impact code quality or run-time/compile-time performance. It is one of the first C++17 features adopted by compiler vendors, and has therefore been made available even on older tool-chains such as vc140. It is more of a cosmetic feature, since it helps improve readability in nested namespace scenarios.
Structured bindings	Needs validation	-
Selection statements with initializer	Needs validation	-
Constexpr if	Needs validation	The feature greatly simplifies compile-time branching that was only possible through overloading or specialization / SFINAE. It improves compile time and readability. The feature is available on all platforms and compilers (VS2017.3)

Utf-8 character literals	Needs validation	-
Direct-list-initialization of enums	Needs validation	-
New standard attributes	Needs validation	The [[nodiscard]] attribute, when put in front of function return type, can prevent discarding important return value or error code or misunderstanding the purpose of function. A typical example is the confusion of .empty() with .clear() in custom containers. With [[nodiscard]] the compiler issues a warning.
Terse static_assert	Needs validation	The terse form of static_assert is useful when the evaluated expression contains enough information that an additional message would be redundant. Supported by all compilers. (VS2017.0) Example: static_assert(std::is_default_constructible_v<T>) is equally readable than static_assert(std::is_default_constructible_v<T>, "Must be default constructible")

Feature	State	Comments
std::variant	Needs validation	(Note: newly supported by PS4 SDK 7.0)
std::optional	Needs validation	(Note: newly supported by PS4 SDK 7.0)
std::any	Disallowed	Not supported on PS4 as of SDK 7.0 when RTTI is turned off.
std::string_view	Needs validation	-
<functional> std::invoke, std::not_fn	Needs validation	-
<tuple> std::apply, std::make_from_tuple	Needs validation	-
<type_traits> std::void_t, std::conjunction, std::disjunction, std::negation, std::invoke_result, std::is_invocable, std::bool_constant		
std::filesystem	Needs validation	-
std::byte	Needs validation	-
Splicing for maps and sets	Needs validation	-
Parallel algorithms	Needs validation	-

Feature	State	Comments
Binary literals	Needs validation	-

Generic lambda expressions	Needs validation	Taking parameters by auto type makes lambda on-par with template functions. It is a great improvement to callbacks and generic algorithms that would be otherwise unnecessarily verbose. It also enables designs such as generic visitor pattern which wasn't possible without the feature. The feature is supported by all compilers.
Lambda capture initializers	Needs validation	-
Return type deduction	Needs validation	-
Decltype (auto)	Needs validation	-
Relaxing constraints on constexpr functions	Needs validation	The feature allows constexpr function to be written in almost identical style as normal run-time functions, improving readability and compile time. Supported by all compilers.
Variable templates	Needs validation	-
[[deprecated]] attribute	Needs validation	-

Feature	State	Comments
User-defined literals for standard library types	Needs validation	-
Compile-time integer sequences	Needs validation	std::integer_sequence and std::index_sequence are useful template-metaprogramming tools that help transform variadic template arguments or constexpr array. Compilers usually implement the feature as magic intrinsics, which speeds up the compilation compared to the manually implemented recursive template. Supported compilers - ?
std::make_unique	Needs validation	-